Configuring a Hierarchical Evolutionary Strategy Using Exploratory Landscape Analysis

Hubert Guzowski
Institute of Computer Science,
AGH University of Kraków
Kraków, Poland
guzowski@agh.edu.pl

Maciej Smołka
Institute of Computer Science,
AGH University of Kraków
Kraków, Poland
smolka@agh.edu.pl

ABSTRACT

Hierarchic Memetic Strategy (HMS) is a stochastic global optimizer designed to tackle highly multimodal problems. It consists of parallel running optimization methods organized in a tree hierarchy. Depending on the task, different algorithms can be utilized on each of the levels. In this paper, we incorporate into HMS's structure a mechanism for choosing its configuration based on information gathered by a set of Exploratory Landscape Analysis (ELA) methods and hyperparametric optimization. We compared the performance of such configured HMS with a portfolio of proven state-of-the-art algorithms on the suite of black-box optimization functions. The results of this work show the efficacy of HMS and provide a set of default parameters evaluated for algorithms users. The use of ELA methods to select the configuration of a composite algorithm extends their standard use as part of an algorithm selector and provides insight into the relationship between exploration and exploitation for different types of fitness functions.

CCS CONCEPTS

• **Theory of computation → Mathematical optimization**.

KEYWORDS

Evolutionary algorithm, Continuous single-objective Optimization, muti-modal optimization, Exploratory Landscape Analysis

ACM Reference Format:
Hubert Guzowski and Maciej Smołka. 2023. Configuring a Hierarchical Evolutionary Strategy Using Exploratory Landscape Analysis. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3583133.3596403

1 INTRODUCTION

The No Free Lunch Theorem for optimization [40] states that average performance on all classes of problems is independent of the algorithm used. Although the original version of the theorem applies to discrete problems without restriction on computational budget, some weaker versions were also proved that show similar behavior in more realistic continuous domain scenarios [2, 8]. Guided by such theorems and supported by experimental results, the field of optimization has focused its efforts on selecting the right algorithm and its parameters for a given problem as defined by John Rice [31]. It is a great task, but approaching it through a lens of meta-learning shows what elements are required to tackle it [35]. One of them is to compare the performance of different optimization algorithms on an extensive set of problem instances. This is done through the curation of representative benchmarks. However, to generalize knowledge beyond the curated cost functions, some measure of problem similarity is needed. Methods devised to calculate numerical features that describe the fitness landscape of a function are referred to under the term exploratory landscape analysis [22]. Based on their results, we can assign problems to specific classes, for which we know the best performing algorithms [23].

Hierarchic Memetic Strategy (HMS) is a development of Hierarchic Genetic Search (HGS), which is a hierarchical stochastic global optimizer first proposed by Schaefer and Kołodziej [34]. HMS was first introduced by Smołka et al. [36] by adding memetic capabilities to the existing HGS structure. HMS is specifically designed to handle difficult real-world optimization problems, especially those related to ill-posed inverse problems with multiple solutions and a very high cost of objective function evaluation. It consists of a set of parallel running genetic processes organized in a tree hierarchy. Processes closer to the root are responsible for exploring the global problem landscape and finding promising regions. Higher-level processes are then sprouted in these regions to perform a more local and precise search. Thus, HMS has built-in mechanisms of adaptivity as it changes its behavior in promising areas. However, by maintaining the exploratory properties of lower-level processes independently of the action of higher ones, we can ensure that the algorithm will not stagnate and will retain an asymptotic guarantee of finding a global optimum [33]. HMS composite structure offers a high level of control over its behavior by choosing algorithms that work at different levels and parameterizing the tree structure itself. However, this level of control comes with a high barrier to entry, especially for users who want to apply HMS out of the box.

In this paper, we enhance the Hierarchic Memetic Strategy by equipping it with a machine-learning-based mechanism to tune its configuration to the given problem. We incorporate into its structure a model that uses Exploratory Landscape Analysis to gather information about the fitness function at a low cost and assign it to a category. The requirement of keeping the computational cost low is directly related to the expected applications of the method in real-world problems, where allowing hundreds of thousands of objective evaluations could mean endless waiting for the result. Simultaneously, we calculated sets of best-performing HMS configurations for each function category, which are then applied to specific problems. Finally, we compared the performance of HMS configured according to our method with state-of-the-art optimization algorithms. Another aspect of our work is to make HMS more accessible. Choosing a proven configuration based on machine learning tools removes a difficult part of the process from the standard user while still leaving space for manual configuration for expert use.

2 RELATED WORKS

Recent years have seen the development of optimization algorithms towards more versatile forms, which are capable of adaptation to a given problem and are better at balancing between exploration and exploitation. Strategies to achieve those improvements vary, but the most prominent are introducing parameter adaptation mechanisms, hybrid algorithms, and selecting from an algorithm portfolio. Adapting algorithms' parameters is often used alongside the two latter approaches, but when applied on its own, it is also capable of producing variants of proven algorithms that outperform their predecessors on a wide range of functions [25, 38]. Hybrid algorithms can be realized by a direct combination of two or more heuristics in a so-called heterogeneous method [13, 21, 26] or by designing a hyperheuristic in which a top-level algorithm selects a locally optimal method [5, 6, 39]. Heterogeneous methods aim to achieve a synergistic relation between their algorithms that overcomes the limitations of their components. Hyperheuristics, on the other hand, are not so focused on specific relations between their components, but rather are a form of an algorithm selection mechanism where the knowledge about the best performing algorithm is gathered during the hyperheuristic execution. Approaches using exploratory landscape analysis methods are similar to hyperheuristics in that they choose from a portfolio of algorithms but differ in that the choice is made once based on the knowledge gained before the proper algorithm execution.

Although some forms of exploratory landscape analysis (ELA) were present much earlier, using it to tackle the algorithm selection problem gained traction with the introduction of inexpensively calculated numerical features [22]. Having a clearly defined problem and a consistent benchmarking environment in the form of the Black-Box Optimization Benchmark (BBOB) [11] meant that over the following years the ELA's collection of tools grew to include new effective methods such as nearest-better clustering [15] and ICoFiS [24]. However, the sources of method implementations remained scattered among isolated repositories and scientific descriptions, making it difficult to use them in practice. To overcome this difficulty, the methods were gathered into a centralized package flacco [18]. Subsequently, Quentin et al. [30] have shown that groups of a few features are sufficient to classify BBOB functions [11] with 98% accuracy. In the same benchmark environment, algorithm selection frameworks based on ELA methods have been shown to be more effective than any single solver in its portfolio [17]. ELA methods are mostly applied for algorithm selection rather than for choosing algorithm configuration. However, the scope of this paper covers both of those aspects, as the complex structure of HMS allows the user to select both the algorithms used on different levels, but also their parameters.

Our work on the HMS algorithm combines efforts to create hybrid metaheuristics and the use of ELA methods in a new context aimed at improving a specific algorithm and making it easier to use. Classical implementations of HMS used the Simple Evolutionary Strategy (SEA) as an evolutionary engine in component processes. In a later work, Sawicki et al. tested the usage of Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [10] in the local phase of HMS, which has been shown to be more efficient in achieving results of the same quality [32]. The flexibility of HMS has been shown in a variety of applications for which its core structure can be tuned. HMS equipped with a multiwinner voting strategy has been shown to be effective in identifying regions of insensitivity of a given problem [9]. Recent work on the algorithm includes its application in neuroevolution [37]. In all of this work, the choice of the HMS configuration was done using a time-consuming trial-and-error approach. Our contribution is meant to alleviate the costs of this process and answer the following set of questions:

- Can we parameterize the HMS based on the classification done using ELA?
- How does automatically configured the HMS compare with other general-purpose global evolutionary search algorithms?

3 DESCRIPTION OF HMS WITH ELA COMPONENT

The core of our HMS implementation does not differ from what was presented in previous works [32]. Here, we summarize the main steps for the convenience of the reader. The HMS core begins at line 4 of Algorithm 1 by sampling a single evolutionary population, becoming the root of the tree. Then, as long as the global stopping condition is not satisfied, every active component population (i.e., a *sub-population*) performs a fixed-length series of evolutionary steps (called a *metaepoch*), i.e. it executes an appropriate evolutionary engine a given number of times, which is abstracted here to a *runMetaepoch()* function call. After a metaepoch, each of the sub-populations can meet a local stopping condition and get *deactivated*. In such a case HMS memorizes the best individual spotted by the sub-population or, if the evolutionary engine is the SEA, it executes a local optimization method with the best found individual as a starting point and memorizes the result (lines 11–13). Then, every sub-population at a non-leaf level tries to *sprout* a child process at the next level of the tree by sampling a population around the parent's current best individual. Afterward, unless the global stopping condition is satisfied, the strategy goes back to running metaepochs in active sub-populations. The final result of the HMS core is the set of individuals memorized by all leaf sub-populations.

A novel aspect of our work is the incorporation of an additional component responsible for choosing HMS configuration shown in lines 1–3 of Algorithm 1. At the start of the algorithm execution, a set of points is generated using Latin hypercube sampling. This set is used as a basis for calculating a set of ELA feature values using implementation provided by the flacco package [18]. These features are an input to a Random Forest classifier that assigns a function to one of the predefined classes for which we calculated the best working configurations beforehand. We decided to use one of the sets of ELA features evaluated in a work by Quentin et al. [30]. Specific aspects of training and testing the classifier and calculating feature sets are described in the following chapter.

Precalculated configurations control both the properties of the heuristics used and the overarching algorithm structure. In our implementation configurations are either two- or three-level trees that use a simple evolutionary algorithm on all levels or swap out the most local processes with CMA-ES. An implementation of the SEA algorithm is based on the LEAP Python package [7] while CMA-ES sub-populations use the Python implementation provided by pyCMA [12]. Structural hyperparameters subjected to optimization are the numbers of generations each sub-population runs during each of the metaepochs, after how many metaepochs they will become inactive, how many processes can be working concurrently on levels higher than root level and how far they can sprout from already existing sub-populations. Parameters specific to SEA are the magnitude of Gaussian mutation and for levels other than root level the variance of normal distribution of initial sample around the sprouting point (best current individual of root). As for the CMA-ES algorithm, the only parameter subject to optimization is the initial sigma (step size) value. Population size in that case is left out to the analytically derived default for this algorithm which is $4 + 3\log(D)$ rounded to the nearest lower integer.

Algorithm 1: High-level pseudocode of HMS with ELA component

```
Input: Objective function $f$, $dimensions$ and $bounds$ of $f$
Output: $foundOptima$ of $f$
1  $ElaFeatureValues \leftarrow$ calculateELA($f$, $dimensions$, $bounds$)
2  $FunctionClass \leftarrow$ classifyFunction($ElaFeatureValues$)
3  $treeConfig \leftarrow$ chooseAndScaleConfig($FunctionClass$, $dimensions$, $bounds$)
4  $root \leftarrow$ doSprout($treeConfig[0]$, $\emptyset$)
5  $activePopulations \leftarrow \{root\}$
6  while global stop condition is not satisfied do
7      foreach $d$ in $activePopulations$ do [in parallel]
8          $isActive \leftarrow$ runMetaepoch($d$)
9          if not $isActive$ then
10             $activePopulations \leftarrow activePopulations \setminus \{d\}$
11             $d.optimum \leftarrow$ bestIndividual($d.history$)
12             if $d.engine = SEA$ then
13                 $d.optimum \leftarrow$ runLocalMethod($d.optimum$)
14     foreach $d$ in $activePopulations$ do [in parallel]
15         if $d$ satisfies sprout condition then
16             $childPopulation \leftarrow$ doSprout($treeConfig[level(d) + 1]$, $d$)
17             $activePopulations \leftarrow activePopulations \cup \{childPopulation\}$
18 $foundOptima \leftarrow []$
19 foreach $d$ in leaf populations do
20     $foundOptima \leftarrow foundOptima \cup \{d.optimum\}$
```

Table 1: Set of ELA features tested in function classification

ELA feature	Source
ϵ_{ratio}	ICoFiS [24]
Fitness correlation	NBC [15]
Adjusted linear regression R2	Meta model [22]
Adjusted quadratic regression R2	Meta model [22]

4 EXPERIMENTAL PROCEDURE

To answer the questions posed in Section 2 a series of consecutive experiments was conducted. Each one tests a different building block of the resulting algorithm. Firstly, we assessed the effectiveness of function classification using ELA with lower evaluation budgets. Secondly, we calculated a set of configurations that perform best for each class of functions and tried to identify trends in the parameter values. Lastly, we compared the effectiveness of the final algorithm in the benchmark environment with state-of-the-art optimization algorithms.

4.1 Function classification

We based the function classification aspect of our work on the research on ELA done in previous studies [22, 24, 30] with an emphasis on a lower evaluation budget. We verified the effectiveness of the set of ELA features presented in table 1 in a benchmark environment. Selection of features was based on a research conducted by Quentin et al. [30]. The benchmark consisted of 24 BBOB functions x [2, 3, 5, 10, 20] dimensions x 25 instances which equals to 3000 different function instances in total. Those functions are divided into five classes with different algorithms that perform best on them [11, 23]. These classes are: separable functions {f1...f5}, functions with low or moderate conditioning {f6...f9}, functions with high conditioning and unimodal {f10...f14}, multimodal functions with adequate global structure {f15...f19}, and multimodal functions with weak global structure {f20...f24}. As described in [3], the significant features of BBOB functions differ from each other even among functions of the same class and therefore are also far apart in the ELA feature space. Generating different instances of those functions helps to alleviate this problem and generalize the results.

We applied ELA methods for above mentioned set of functions using package flacco [18], Latin-Hypercube sampling and evaluation budgets of 15, 25, 50, 100 and 200 individuals per function dimension respectively. We divided the resulting feature sets using 80/20 ratio into training and test sets and then used to train Random Forest classifiers. For consistent results, we repeated splitting the feature set and training classification model 50 times per each evaluation budget.

4.2 Calculation of default parameters

Hyperparameter optimization task was assigned to the SMAC3 optimization tool [19]. The optimization process was run with a budget of 10^3 hyperparameter evaluations and each one of them was repeated 2 times during optimization to better account for the stochastic nature of the process. Each evaluation corresponded to performing an optimization process on all of the functions in specific BBOB class with a budget of 10^4 evaluations for each one of them. It is worth noting that the median number of evaluations to achieve results deemed as successful of all the algorithms in the BBOB data is about 10^5 [3]. The chosen budget is meant to reflect the usage of HMS in costly real-world problems. Configurations were evaluated by their quality calculated as a sum of best found optima for each of optimized functions. We tested three HMS structures in that manner: two-level tree with SEA at both levels, two-level tree with CMA-ES used on second level instead and a three-level tree with SEA at first two levels and CMA-ES at third level. For each BBOB function class the whole optimization process was run 5 times per each of three chosen HMS configurations giving the total of 75 resulting hyperparameter sets.

Having obtained the 75 configurations, we performed additional selection based on another 50 optimization runs for each of the configurations for each of the functions belonging to its class. We selected one of the HMS structures based on the number of functions for which their representatives achieved the best average results. After that we selected a specific configuration among the five of that general structure based on the same criteria. The result is a single configuration for each of the function classes.

4.3 Comparison of algorithms performance

The selection of a representative set of algorithms with which we want to compare the new method is not obvious. In our selection, we kept in mind that the selected algorithms should be well tested and give the best results for different classes of functions. The availability of the algorithm implementation was also important to us. In the end, we decided to include in the comparative portfolio 2 variants of the Covariance Matrix Adaptation Evolution Strategy algorithm, a variant of Differential Evolution and the NEWUOA method. CMA-ES with Diagonal Acceleration (dd-CMA-ES) [1] is one of the newer variants of this well-established algorithm, which performs better with separable functions (class 1) without losing efficiency for the other classes. We have included restarts with increasing population size for this variant. The second variant we used is bipop-CMA-ES [20], which performs restarts with populations of different sizes, improving the algorithm's performance for multimodal functions (classes 4 and 5). The implementation we used for both of these variants is available in the pycma library [12]. The Differential Evolution variant we used is an improved Success-History based Adaptive DE with the linear population size reduction mechanism (iL-SHADE) [4]. It gives better results than its original, which was the best-rated DE variant. iL-SHADE is very effective for multimodal functions, which is well complemented by the choice of the CMA-ES algorithm. The implementation of iL-SHADE that we used is available in the pyade library [29] The last algorithm compared is NEWUOA [27], which does very well with the average evaluation budget for highly dimensional functions. Despite being developed in 2006, NEWUOA continues to perform very well compared to the new algorithms for unimodal functions in particular. We used the implementation of NEWUOA available in pdfo library [28]. We also considered using SMAC-BBOB [14] for comparison, which also has competetive advantages in benchmark environment, but as the authors of the algorithm noted, SMAC-BBOB is overperformed by CMA-ES somewhere between $10D$ and $100D$ evaluations and we use $100D$ evaluations in our comparison (D is the problem dimensionality).

We benchmark HMS against the presented portfolio of algorithms and present the results obtained both when using only configurations assigned to a specific class and chosen accordingly to the accuracy of classification of a machine learning model tested in previous steps. We run each algorithm 50 times on first instance of each function included in BBOB continuous optimization benchmark with a total budget of 10^4 evaluations. Resulting best found fitnesses are then normalized by subtracting the value of known global optimum. We did not implement any mechanism that would cut off errors lower than some specific precision, instead relying on the precision of Python floating-point arithmetic and mechanisms specific to libraries used. To properly reflect the cost of parameter selection process, evaluations used for choosing a configuration were subtracted from the budget of HMS. To additionaly test if the configurations obtained for HMS during hyperoptimization process could generalize from the original 10-dimensional test bed, we performed optimization both in 10 and 20 dimensions without changing or upscaling the parameters (with the exception of CMA-ES sub-populations, which used default population values of that algorithm, which have built-in scaling). To ensure that HMS does not exceed its evaluation budget during runtime, we implemented a wrapper over BBOB functions that counted the number of evaluations and when exceeding it, returned positive infinity instead of a function value. For other functions we relied on mechanisms included in their implementation.

5 RESULTS

This section presents the results of the experiments carried out in subsections which correspond to their descriptions in Section 4.

5.1 Classification results

Our results do not show a significant decrease in quality when using 50 evaluations per dimension when using the feature set presented in table 1. See figure 1. The results are in line with conclusions derived by Kerschke et al. when testing the effectiveness of ELA on lower budgets [16]. However, most of the works use a number of individuals higher than 100 individuals per function dimension [3, 22, 24]. Note that in following benchmarks we use budgets of 10^4 order of magnitude while the median of expected number of evaluations to achieve results deemed as succesful at BBOB competitions is around 10^5 so such an upfront cost although justifiable in other context, cannot be amortized in our application. Thus we decided on using a classifier trained on 50 evaluations per function dimension for HMS.

Accuracy achieved for each of the BBOB classes is presented in figure 2. We achieved an accuracy rate of 91,89% for all combined classes. Separable and multimodal functions with weakly defined structure were classified with highest accuracy which is in line with intuition that those classes are the most distinct. Classes 2, 3 and 4 were mainly incorrectly classified within these three classes. At the same time, the lower scores for those three classes were not evenly distributed among their functions. In particular, instances 7, 14, 17 and 18 were relatively often assigned incorrectly while errors were rather rare in the other instances. These errors may be

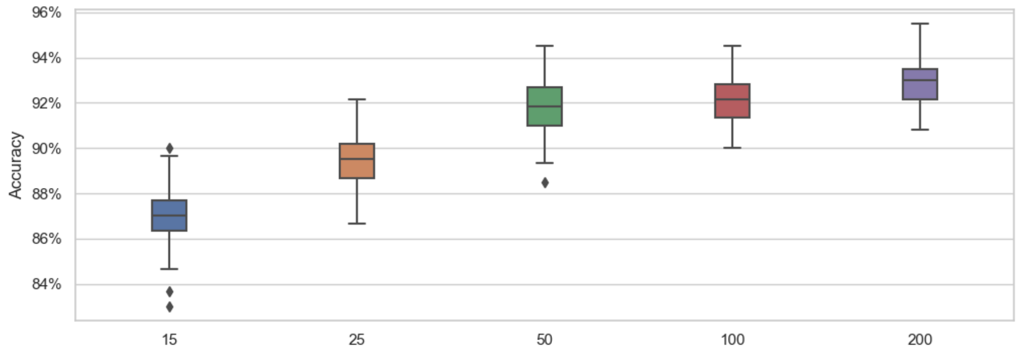

Figure 1: Classification accuracy per number of evaluations per dimension in initial sample

Table 2: Confusion matrix for Random Forest classifier trained on ELA results

		Output class					Total rates of success/failure	
		{f1...f5}	{f6...f9}	{f10...f14}	{f15...f19}	{f20...f24}		
Target class	{f1...f5}	20%	0,097%	0,18%	0,14%	0.0%	97,9%	2,0%
	{f6...f9}	0,35%	15%	0,26%	0,99%	0,04%	89,8%	10,2%
	{f10...f14}	0,65%	0,72%	18%	1,5%	0,0%	86,4%	13,6%
	{f15...f19}	0,21%	0,68%	1,0%	18%	0,65%	87,6%	12,4%
	{f20...f24}	0%	0,15%	0%	0,48%	20%	97,0%	3,0%
Total rates of success/failure		94,3%	89,8%	86,4%	87,6%	97,0%	91,89%	
		5,7%	10,2%	13,6%	12,4%	3,0%		8,11%

Table 3: Tree structure and parameters of algorithm controlling the proportion of budget for first and second level of HMS algorithm for different function classes. The cma caption in the column indicates the use of the default population size for the CMA-ES algorithm, which is 10 and 12 for 10 and 20-dimensional functions, respectively.

		Level 1		Level 2		
function set	choosen configuration	population	generations	population	generations	upper limit of processes
{f1...f5}	2 level with cma leafs	22	2	cma	8	3
{f6...f9}	2 level with sea leafs	29	3	20	2	10
{f10...f14}	2 level with cma leafs	21	2	cma	10	6
{f15...f19}	2 level with cma leafs	25	2	cma	9	5
{f20...f24}	2 level with sea leafs	78	2	35	6	4

due to a defect in the classifier, or rather, they may indicate certain properties of the functions as appearing on a spectrum between classes rather than belonging to any particular one. Although, even those function instances were most often classified as their true BBOB class. However, what is the impact of misclassification and consequent use of a different set of parameters, will only be shown in subsequent tests.

5.2 Resulting default parameters

The results of the parameter selection process give us an insight into how global optimization algorithms work and how flexibly the HMS algorithm can be adapted to different tasks. In this section, we have chosen to concisely include what we believe are the most important elements of the setup that provide the most interesting conclusions.

Among the 3 configurations tested, the three-level HMS tree did not give the best results for almost any function. Among the two two-level structures, we could clearly observe better results for one of the categories in the case of classes 1, 4 and 5, where the ratio of the number of functions for which the structure gave the best results was 7:3 for class 1 and 4, respectively, while for class 5 in each case better results were achieved when CMA-ES was used in the algorithm. For class 2 and 3, the ratios were 3:5 and 4:6, respectively. The configurations selected based on the number of functions for which they gave the best results are listed in column 2 of table 3.

Table 3 shows what population sizes and generations per metaepoch were set for the best configurations for each class. The ratio of the products of these two values for the two levels translates directly into the proportion of the budget allocated to the operation of both of them. We can, according to the assumptions of the HMS algorithm, equate this ratio with the relationship between the property of exploration and exploitation of the algorithm. This interpretation coincides with the results obtained and confirms the possibility of adjusting the behavior of the HMS algorithm in this way. As we can see in table 3, for function class 5, the parameters obtained direct the HMS operation to the highest degree towards exploration, which is consistent with the multimodality and weak global structure of these functions. The configurations of classes 1, 3 and 4 do not differ significantly in the number of generations and population sizes, but for the first class we can observe in this case a clearly smaller upper limit of parallel processes at the second level. Such a value, in our interpretation, corresponds to the unimodal and separable nature of these functions, which in effect makes more processes at the second level unnecessary, since they would occupy the same optimum with high probability.

5.3 Comparison results

We tested HMS against dd-CMA-ES [1], BIPOP-CMA-ES [20], iL-SHADE [4] and NEWUOA [27] on 10 and 20-dimensional first instances of Black Box Optimization noiseless continuous test suite. We include two sets of results for HMS. First column of table 4 shows the effectiveness of the algorithm when configured using the set of parameters calculated for its BBOB function class. Second column takes into account the inaccuracy of the classifier and includes weigted average and variance of results obtained in first column and those obtained using the configurations calculated for other function classes in proportions corresponding to the rates of classification to those classes as calculated in first experiment 4.2. Out of all 48 functions included in our test suite misclassifications appeared in 17. Out of those 17, in 12 functions we observed the loss in quality of solutions due to those errors. Having in mind, that in practice the algorithm operates on a singular classifier and that each of the presented functions is most often classified as its true BBOB class, we treat the results from the first column as representative for HMS performance.

We have highlighted the best results achieved for each function in table 4. We exempted the second column from that comparison as mostly duplicates the results from the first column and as mentioned, its purpose is rather to show theoretical expected values, when adjusted for the innacuracy of the classifier. HMS produced the best or equal best results in 19 out of the 48 test cases. According to its purpose, it performed better for multimodal functions than unimodal ones, for which maintaining exploration throughout the algorithm's operation is not as important. At the same time it achieved at least one best or equal best result for every one of 5 function classes. Overall HMS performs on par with compared state-of-the-art methods on most of the problems included in our test suite which shows its flexibility when parameterized accordingly. Still there are function instances, for which HMS performs considerable worse than other algorithms. Those are specifically functions characterized by a better defined unimodal structure, for which other algorithms are able to exploit the minimas better, than HMS. In those cases the cost connected with maintaining exploratory qualities of root node and simultaneous executions of more than one sub-population at lower levels makes it less likely that HMS will locate the minima as precisely as the algorithms focused more on exploitation for the same evaluation budget. Interestingly, HMS did better when applied to 20-dimensional problems despite the fact that the hyperparameter optimization process was performed solely on 10-dimensional functions. Higher-dimensional problems are harder to solve, which is manifested in the overall worse performance of optimization algorithms, but HMS and iL-SHADE scale quite well to tackle those problems and are more likely to outperform variants of CMA-ES. These results reinforce our assumption about the use of HMS to solve difficult multi-modal problems.

6 CONCLUSIONS

In this paper, we introduced a machine-learning mechanism for selecting a configuration of Hierarchic Memetic Strategy algorithm based on values derived using Exploratory Landscape Analysis. We selected a set of ELA features, calculated their values on a suite of functions from Black Box Optimization Benchmark and used them to train a classifier which assigns a problem to one of five classes distinguished in BBOB benchmark. We performed hyperparametric optimization to derive the best performing configurations of HMS for each of those classes and finally compared the resulting algorithm with a portfolio of state-of-the-art optimization algorithms on a moderate evaluation budget.

The classification results obtained and the fitness function values obtained for the test functions confirm the validity of using a machine learning model to determine HMS configurations. The results obtained in this way are on par with state-of-the-art algorithms for each class of functions and for multimodal functions with weak global structure, the competitive advantage of HMS is apparent. At the same time, the obtained parameter sets are in line with the intuition on the relationship between exploration and exploitation for different classes of functions. This shows the flexibility of the HMS mechanism and the adaptivity of its structure to features of considered problems.

The current results are promising although they are achieved in a benchmark setting under specific conditions. In future works we plan to further prove the capabilities of configuration selection mechanism by performing more robust tests and applying the method to real-world optimization problems.

ACKNOWLEDGMENTS

The research presented in this paper was partially supported by the Polish National Science Center under grant No. 2020/39/I/ST7/02285 and by the funds of the Polish Ministry of Science and Education assigned to the AGH University of Krakow.

REFERENCES

[1] Y. Akimoto and N. Hansen. 2020. Diagonal Acceleration for Covariance Matrix Adaptation Evolution Strategies. *Evolutionary Computation* 28, 3 (09 2020), 405–435. https://doi.org/10.1162/evco_a_00260 arXiv:https://direct.mit.edu/evco/article-pdf/28/3/405/1858973/evco_a_00260.pdf

[2] Anne Auger and Olivier Teytaud. 2010. Continuous Lunches Are Free Plus the Design of Optimal Optimization Algorithms. *Algorithmica* 57, 1 (01 May 2010), 121–146. https://doi.org/10.1007/s00453-008-9244-5

[3] Bernd Bischl, Olaf Mersmann, Heike Trautmann, and Mike Preuss. 2012. Algorithm Selection Based on Exploratory Landscape Analysis and Cost-Sensitive Learning. In *Proceedings of the 14th International Conference on Genetic and*

Table 4: Comparison results between HMS with calculated configurations, adjusted for classification errors and algorithms in portfolio. Each column contains average value of fitness and its variance in brackets.

	HMS	adjusted HMS	dd-CMA-ES	BIPOP-CMA-ES	iL-SHADE	NEWUOA
	10 dimensions					
f1	9.4e-15 (7.8e-15)	9.4e-15 (7.8e-15)	4.3e-16 (2.4e-15)	1.2e-14 (9.6e-15)	1.1e-14 (1.2e-14)	**0.0e+00** (0.0e+00)
f2	7.7e-01 (1.2e+00)	7.6e-01 (1.2e+00)	**0.0e+00** (0.0e+00)	1.6e-14 (2.6e-14)	1.3e-11 (3.4e-11)	1.2e+00 (1.5e+00)
f3	8.4e+00 (3.1e+00)	8.4e+00 (3.1e+00)	**4.6e+00** (2.0e+00)	7.8e+00 (3.7e+00)	1.1e+01 (3.7e+00)	1.3e+02 (6.8e+01)
f4	1.4e+01 (4.7e+00)	1.4e+01 (4.7e+00)	**1.0e+01** (2.8e+00)	1.1e+01 (3.8e+00)	1.2e+01 (3.6e+00)	1.8e+02 (1.3e+02)
f5	6.8e-15 (2.4e-14)	6.8e-15 (2.4e-14)	**0.0e+00** (0.0e+00)	**0.0e+00** (0.0e+00)	1.1e-11 (5.1e-11)	**0.0e+00** (0.0e+00)
f6	7.2e-01 (3.7e+00)	7.2e-01 (3.7e+00)	6.0e-13 (5.5e-13)	**5.0e-13** (3.2e-13)	3.2e-05 (5.0e-05)	1.3e-04 (8.1e-04)
f7	1.1e+00 (6.5e-01)	1.0e+00 (6.1e-01)	**1.9e-14** (2.6e-14)	1.8e-13 (1.3e-12)	2.8e-03 (2.6e-02)	8.2e+01 (7.5e+01)
f8	**1.2e-08** (1.1e-08)	1.2e-08 (1.1e-08)	1.9e-01 (8.0e-01)	1.1e-01 (4.9e-01)	2.7e+00 (1.2e+00)	8.0e-01 (1.6e+00)
f9	**3.7e-08** (3.7e-08)	3.7e-08 (3.7e-08)	1.1e-01 (5.5e-01)	9.7e-02 (4.7e-01)	4.0e+00 (1.2e+00)	1.1e+00 (1.8e+00)
f10	1.0e+00 (6.8e+00)	1.0e+00 (6.8e+00)	**1.5e-14** (1.4e-14)	1.7e-14 (1.4e-14)	3.0e-01 (1.7e+00)	1.3e+00 (1.8e+00)
f11	1.5e+00 (5.9e+00)	1.5e+00 (5.9e+00)	**1.2e-14** (7.4e-15)	**1.2e-14** (8.4e-15)	3.1e-02 (2.3e-01)	3.0e-02 (5.1e-02)
f12	1.8e+00 (2.6e+00)	1.8e+00 (2.6e+00)	**2.3e-05** (1.9e-04)	1.2e-04 (8.8e-04)	6.6e-01 (9.0e-01)	6.6e-03 (2.7e-02)
f13	7.8e-03 (2.9e-02)	7.8e-03 (2.9e-02)	1.9e-06 (1.4e-05)	**5.8e-07** (2.2e-06)	6.1e-03 (1.1e-02)	7.2e+00 (8.9e+00)
f14	1.1e-11 (9.4e-12)	1.2e-11 (9.8e-12)	9.8e-12 (6.1e-12)	**8.5e-12** (5.0e-12)	2.4e-06 (4.6e-06)	5.7e-06 (1.5e-06)
f15	1.0e+01 (5.2e+00)	1.0e+01 (5.2e+00)	6.3e+00 (3.7e+00)	**6.1e+00** (2.9e+00)	1.2e+01 (3.8e+00)	1.9e+02 (1.1e+02)
f16	**3.0e-01** (3.7e-01)	5.0e-01 (4.1e-01)	5.2e-01 (5.4e-01)	3.8e-01 (6.5e-01)	2.0e+00 (1.3e+00)	1.7e+01 (9.9e+00)
f17	9.5e-02 (1.7e-01)	1.8e-01 (2.3e-01)	9.1e-02 (2.8e-01)	4.4e-02 (2.6e-01)	**3.0e-04** (3.7e-04)	1.7e+01 (1.7e+01)
f18	6.4e-01 (1.1e+00)	1.1e+00 (1.3e+00)	2.8e-01 (5.3e-01)	9.1e-02 (3.6e-01)	**1.1e-02** (5.9e-02)	7.1e+01 (7.8e+01)
f19	2.1e+00 (1.4e+00)	2.1e+00 (1.4e+00)	1.1e+00 (7.7e-01)	**1.0e+00** (7.4e-01)	1.9e+00 (4.7e-01)	7.3e+00 (4.7e+00)
f20	**1.2e+00** (2.3e-01)	1.2e+00 (2.3e-01)	**1.2e+00** (3.0e-01)	1.4e+00 (2.8e-01)	1.5e+00 (2.5e-01)	1.6e+00 (3.5e-01)
f21	1.4e+00 (1.4e+00)	1.4e+00 (1.4e+00)	1.4e+00 (1.5e+00)	**1.0e+00** (9.8e-01)	3.5e+00 (5.1e+00)	1.7e+01 (1.7e+01)
f22	**1.3e+00** (8.6e-01)	1.3e+00 (8.6e-01)	2.5e+00 (1.0e+00)	2.4e+00 (5.8e-01)	2.7e+00 (1.2e+00)	2.1e+01 (2.1e+01)
f23	1.0e+00 (4.4e-01)	1.0e+00 (4.4e-01)	**9.6e-01** (7.1e-01)	1.1e+00 (7.2e-01)	1.5e+00 (3.0e-01)	1.5e+00 (8.4e-01)
f24	**2.0e+01** (5.8e+00)	2.0e+01 (5.8e+00)	2.2e+01 (1.0e+01)	2.2e+01 (9.4e+00)	2.9e+01 (8.3e+00)	2.1e+02 (8.1e+01)
	20 dimensions					
f1	1.6e-14 (4.6e-15)	1.6e-14 (4.6e-15)	9.2e-15 (6.8e-15)	1.9e-14 (8.6e-15)	1.4e-05 (1.6e-05)	**0.0e+00** (0.0e+00)
f2	7.4e+01 (3.4e+01)	7.4e+01 (3.4e+01)	**4.8e-14** (2.1e-14)	9.4e+00 (2.2e+01)	9.7e-01 (2.4e+00)	1.5e+01 (1.6e+01)
f3	2.7e+01 (7.0e+00)	2.7e+01 (7.0e+00)	**1.8e+01** (5.4e+00)	3.3e+01 (1.3e+01)	7.3e+01 (1.2e+01)	3.9e+02 (2.0e+02)
f4	3.9e+01 (1.0e+01)	3.9e+01 (1.0e+01)	**2.9e+01** (8.4e+00)	4.3e+01 (1.3e+01)	8.9e+01 (1.2e+01)	4.5e+02 (2.1e+02)
f5	**8.5e-14** (0.0e+00)	8.5e-14 (0.0e+00)	**8.5e-14** (0.0e+00)	**8.5e-14** (0.0e+00)	2.7e-02 (2.4e-02)	**8.5e-14** (0.0e+00)
f6	7.8e+02 (5.4e+06)	6.1e+02 (1.3e+03)	1.4e-07 (2.2e-07)	**8.3e-08** (3.0e-07)	2.5e+00 (1.1e+00)	5.3e-07 (5.1e-06)
f7	8.8e+00 (1.1e+01)	9.1e+00 (2.9e+00)	1.5e+00 (9.7e-01)	1.9e+00 (2.9e+00)	**1.1e+00** (8.2e-01)	2.2e+02 (2.9e+02)
f8	**2.5e-08** (2.4e-16)	2.8e-08 (1.5e-08)	2.7e+00 (1.9e+00)	6.2e+00 (7.0e+00)	1.6e+01 (1.3e+00)	6.0e-01 (1.4e+00)
f9	**8.4e-08** (3.2e-15)	7.0e-08 (4.3e-08)	4.8e+00 (2.3e+00)	5.3e+00 (2.3e+00)	1.7e+01 (8.6e-01)	1.2e+00 (1.8e+00)
f10	**7.2e+00** (1.8e+01)	7.2e+00 (1.8e+01)	1.1e+01 (2.9e+01)	9.2e+00 (2.0e+01)	4.9e+02 (3.5e+02)	1.9e+01 (2.1e+01)
f11	2.3e+00 (7.9e+00)	2.3e+00 (7.9e+00)	2.3e-14 (2.0e-14)	**2.1e-14** (9.9e-15)	5.7e+00 (3.6e+00)	3.6e+01 (4.6e+01)
f12	**2.2e-01** (8.6e-01)	2.2e-01 (8.6e-01)	1.4e+00 (3.6e+00)	1.9e+00 (9.8e+00)	3.1e+01 (5.4e+01)	3.4e-01 (1.3e+00)
f13	**2.5e-01** (6.7e-01)	2.6e-01 (6.7e-01)	3.2e-01 (5.7e-01)	4.6e-01 (8.1e-01)	3.8e+00 (2.7e+00)	7.4e+00 (1.1e+01)
f14	**3.9e-09** (3.1e-09)	8.5e-07 (3.0e-06)	2.8e-07 (1.7e-07)	2.9e-07 (1.7e-07)	1.8e-03 (9.2e-04)	1.8e-05 (1.8e-06)
f15	**2.5e+01** (7.3e+00)	2.7e+01 (7.8e+00)	2.8e+01 (1.1e+01)	3.3e+01 (1.0e+01)	8.7e+01 (1.2e+01)	4.5e+02 (1.7e+02)
f16	**5.1e-01** (4.0e-01)	1.5e+00 (5.0e-01)	4.5e+00 (2.7e+00)	4.0e+00 (2.9e+00)	1.1e+01 (2.7e+00)	2.2e+01 (1.1e+01)
f17	**3.4e-02** (5.8e-02)	5.2e-02 (6.9e-02)	4.9e-01 (5.5e-01)	1.4e-01 (2.7e-01)	7.3e-02 (6.6e-02)	1.4e+01 (8.7e+00)
f18	4.6e-01 (7.2e-01)	7.4e-01 (8.2e-01)	1.7e+00 (2.2e+00)	6.8e-01 (9.4e-01)	**4.4e-01** (2.5e-01)	6.0e+01 (2.7e+01)
f19	4.2e+00 (1.6e+00)	4.2e+00 (1.6e+00)	3.1e+00 (1.7e+00)	**3.3e+00** (1.6e+00)	4.3e+00 (4.3e-01)	9.4e+00 (4.1e+00)
f20	**1.5e+00** (1.5e-01)	1.5e+00 (1.5e-01)	1.6e+00 (2.2e-01)	1.7e+00 (2.2e-01)	2.6e+00 (1.5e-01)	1.7e+00 (3.0e-01)
f21	**1.9e+00** (2.2e+00)	1.9e+00 (2.2e+00)	2.6e+00 (2.6e+00)	2.4e+00 (2.7e+00)	5.7e+00 (7.5e-01)	1.3e+01 (1.5e+01)
f22	3.3e+00 (4.4e+00)	3.4e+00 (4.5e+00)	5.5e+00 (1.0e+01)	5.2e+00 (8.8e+00)	**5.5e-03** (1.2e-02)	2.6e+01 (2.5e+01)
f23	**1.5e+00** (5.0e-01)	1.5e+00 (5.0e-01)	2.2e+00 (1.2e+00)	2.5e+00 (1.0e+00)	2.6e+00 (4.0e-01)	1.7e+00 (9.3e-01)
f24	6.9e+01 (1.3e+01)	6.9e+01 (1.3e+01)	**6.1e+01** (3.4e+01)	7.7e+01 (4.2e+01)	1.1e+02 (1.5e+01)	5.3e+02 (1.3e+02)

Evolutionary Computation. Association for Computing Machinery, New York, NY, USA, 313–320. https://doi.org/10.1145/2330163.2330209

[4] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. 2016. iL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, Vancouver, Canada, 1188–1195. https://doi.org/10.1109/CEC.2016.7743922

[5] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. 2019. A Classification of Hyper-Heuristic Approaches: Revisited. In *Handbook of Metaheuristics*, Michel Gendreau and Jean-Yves Potvin (Eds.). Springer International Publishing, Cham, 453–477. https://doi.org/10.1007/978-3-319-91086-4_14

[6] Olacir R. Castro, Gian Mauricio Fritsche, and Aurora Pozo. 2018. Evaluating selection methods on hyper-heuristic multi-objective particle swarm optimization. *Journal of Heuristics* 24, 4 (01 08 2018), 581–616. https://doi.org/10.1007/s10732-018-9369-x

[7] Mark A. Coletti, Eric O. Scott, and Jeffrey K. Bassett. 2020. Library for Evolutionary Algorithms in Python (LEAP). In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 1571–1579. https://doi.org/10.1145/3377929.3398147

[8] Stefan Droste, Thomas Jansen, and Ingo Wegener. 2002. Optimization with randomized search heuristics—the (A)NFL theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science* 287, 1 (2002), 131–144. https://doi.org/10.1016/S0304-3975(02)00094-4 Natural Computing.

[9] Piotr Faliszewski, Jakub Sawicki, Robert Schaefer, and Maciej Smolka. 2017. Multiwinner Voting in Genetic Algorithms. *IEEE Intelligent Systems* 32, 1 (2017), 40–48. https://doi.org/10.1109/MIS.2017.5

[10] Nikolaus Hansen. 2005. The CMA Evolution Strategy: A Tutorial. https://hal.inria.fr/hal-01297037 ArXiv e-prints, arXiv:1604.00772, 2016, pp.1-39.

[11] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup*. Research Report RR-6828. INRIA. https://hal.inria.fr/inria-00362649

[12] Nikolaus Hansen, yoshihikoueno, ARF1, Gabriela Kadlecová, Kento Nozawa, Luca Rolshoven, Matthew Chan, Youhei Akimoto, brieglhostis, and Dimo Brockhoff. 2023. *CMA-ES/pycma: r3.3.0*. GitHub. https://doi.org/10.5281/zenodo.7573532

[13] Kyle Robert Harrison, Andries P. Engelbrecht, and Beatrice M. Ombuki-Berman. 2017. An adaptive particle swarm optimization algorithm based on optimal parameter regions. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, Honolulu, HI, USA, 1–8. https://doi.org/10.1109/SSCI.2017.8285342

[14] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2013. An Evaluation of Sequential Model-Based Optimization for Expensive Blackbox Functions. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation* (Amsterdam, The Netherlands) *(GECCO '13 Companion)*. Association for Computing Machinery, New York, NY, USA, 1209–1216. https://doi.org/10.1145/2464576.2501592

[15] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. 2015. Detecting Funnel Structures by Means of Exploratory Landscape Analysis. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (Madrid, Spain) *(GECCO '15)*. Association for Computing Machinery, New York, NY, USA, 265–272. https://doi.org/10.1145/2739480.2754642

[16] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. 2016. Low-Budget Exploratory Landscape Analysis on Multiple Peaks Models. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (Denver, Colorado, USA) *(GECCO '16)*. Association for Computing Machinery, New York, NY, USA, 229–236. https://doi.org/10.1145/2908812.2908845

[17] Pascal Kerschke and Heike Trautmann. 2019. Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning. *Evolutionary Computation* 27, 1 (3 2019), 99–127. https://doi.org/10.1162/evco_a_00236

[18] Pascal Kerschke and Heike Trautmann. 2019. Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-package flacco. In *Applications in Statistical Computing – From Music Data Analysis to Industrial Quality Improvement*, Nadja Bauer, Katja Ickstadt, Karsten Lübke, Gero Szepannek, Heike Trautmann, and Maurizio Vichi (Eds.). Springer, Cham, 93–123. https://doi.org/10.1007/978-3-030-25147-5_7

[19] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23, 54 (2022), 1–9. http://jmlr.org/papers/v23/21-0888.html

[20] Ilya Loshchilov, Marc Schoenauer, and Michele Sèbag. 2013. Bi-Population CMA-ES Agorithms with Surrogate Models and Line Searches. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation* (Amsterdam, The Netherlands) *(GECCO '13 Companion)*. Association for Computing Machinery, New York, NY, USA, 1177–1184. https://doi.org/10.1145/2464576.2482696

[21] Jacek Mańdziuk and Adam Żychowski. 2016. A memetic approach to vehicle routing problem with dynamic requests. *Applied Soft Computing* 48 (2016), 522–534. https://doi.org/10.1016/j.asoc.2016.06.032

[22] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory Landscape Analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Dublin, Ireland) *(GECCO '11)*. Association for Computing Machinery, New York, NY, USA, 829–836. https://doi.org/10.1145/2001576.2001690

[23] Olaf Mersmann, Mike Preuss, and Heike Trautmann. 2010. Benchmarking Evolutionary Algorithms: Towards Exploratory Landscape Analysis. In *Parallel Problem Solving from Nature, PPSN XI*, Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 73–82.

[24] Mario A. Muñoz, Michael Kirley, and Saman K. Halgamuge. 2015. Exploratory Landscape Analysis of Continuous Space Optimization Problems Using Information Content. *IEEE Transactions on Evolutionary Computation* 19, 1 (2015), 74–87. https://doi.org/10.1109/TEVC.2014.2302006

[25] Duc Manh Nguyen. 2018. An Adapting Population Size Approach in the CMA-ES for Multimodal Functions. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 219–220. https://doi.org/10.1145/3205651.3205801

[26] Michał Okulewicz, Mateusz Zaborski, and Jacek Mańdziuk. 2022. Self-Adapting Particle Swarm Optimization for continuous black box optimization. *Applied Soft Computing* 131 (2022), 109722. https://doi.org/10.1016/j.asoc.2022.109722

[27] M. J. D. Powell. 2006. The NEWUOA software for unconstrained optimization without derivatives. In *Large-Scale Nonlinear Optimization*, G. Di Pillo and M. Roma (Eds.). Springer US, Boston, MA, 255–297. https://doi.org/10.1007/0-387-30065-1_16

[28] T. M. Ragonneau and Z. Zhang. 2023. PDFO: a cross-platform package for Powell's derivative-free optimization solvers. arXiv:2302.13246.

[29] David Criado Ramón. 2023. *PyADE: Python Advanced Differential Evolution*. GitHub. https://github.com/xKuZz/pyade

[30] Quentin Renau, Johann Dreo, Carola Doerr, and Benjamin Doerr. 2021. Towards Explainable Exploratory Landscape Analysis: Extreme Feature Selection for Classifying BBOB Functions. In *Applications of Evolutionary Computation*, Pedro A. Castillo and Juan Luis Jiménez Laredo (Eds.). Springer International Publishing, Cham, 17–33.

[31] John R. Rice. 1976. The Algorithm Selection Problem. *Advances in Computers* 15 (1976), 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

[32] Jakub Sawicki, Marcin Łoś, Maciej Smołka, and Julen Alvarez-Aramberri. 2019. Using Covariance Matrix Adaptation Evolutionary Strategy to boost the search accuracy in hierarchic memetic computations. *Journal of Computational Science* 34 (2019), 48–54. https://doi.org/10.1016/j.jocs.2019.04.005

[33] Jakub Sawicki, Marcin Łoś, Maciej Smołka, and Robert Schaefer. 2022. Understanding Measure-Driven Algorithms Solving Irreversibly Ill-Conditioned Problems. *Natural Computing* 21 (6 2022), 289—-315. https://doi.org/10.1007/s11047-020-09836-w

[34] Robert Schaefer and Joanna Kołodziej. 2002. Genetic Search Reinforced by the Population Hierarchy. In *Proceedings of the Seventh Workshop on Foundations of Genetic Algorithms, September 2-4, 2002*, Kenneth De Jong, Riccardo Poli, and Jonathan E. Rowe (Eds.). Morgan Kaufmann, Torremolinos, Spain, 383–400.

[35] Kate Smith-Miles. 2008. Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection. *ACM Comput. Surv.* 41 (12 2008). https://doi.org/10.1145/1456650.1456656

[36] Maciej Smołka, Robert Schaefer, Maciej Paszyński, David Pardo, and Julen Álvarez-Aramberri. 2015. An Agent–Oriented Hierarchic Strategy for Solving Inverse Problems. *International Journal of Applied Mathematics and Computer Science* 25, 3 (2015), 483–498. https://doi.org/10.1515/amcs-2015-0036

[37] Mateusz Sokół and Maciej Smołka. 2022. Application of the Hierarchic Memetic Strategy HMS in Neuroevolution. In *Computational Science – ICCS 2022*. Springer, Cham, 422–429. https://doi.org/10.1007/978-3-031-08754-7_49

[38] Ryoji Tanabe and Alex Fukunaga. 2014. Improving the Search Performance of SHADE Using Linear Population Size Reduction. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*. IEEE, Beijing, China, 1658–1665. https://doi.org/10.1109/CEC.2014.6900380

[39] Stefan A.G. van der Stockt and Andries P. Engelbrecht. 2018. Analysis of selection hyper-heuristics for population-based meta-heuristics in real-valued dynamic optimization. *Swarm and Evolutionary Computation* 43 (2018), 127–146. https://doi.org/10.1016/j.swevo.2018.03.012

[40] D.H. Wolpert and W.G. Macready. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 67–82. https://doi.org/10.1109/4235.585893